APPENDIX A

```
/*
******************************** LBUCKET.C ********************************
 *
 *
 *                       Copyright (c) 1998
 *                          OLICOM A/S
 *                            Denmark
 *
 *                      All Rights Reserved
 *
 *  This source file is subject to the terms and conditions of the
 *  OLICOM Software License Agreement which restricts the manner
 *  in which it may be used.
 *
 * --------------------------------------------------------------------------
 *
 * Description: Leaky bucket routines
 *
 * --------------------------------------------------------------------------
 *
 * $Log: lbucket.c.v $
 * Revision 1.3  1998/06/15 12:31:45  cto
 * CreditLeft now returns sdword rather than sword
 *
 * Revision 1.2  1998/06/15 11:25:14  cto
 * Modifications for process overload buckets.
 * Function UseLeakyBucketAfter added.
 *
 * Revision 1.1  1998/06/04 10:43:06  cto
 * Addition of leaky bucket files lbucket.c, lbucket.h
 *
 *
 *
 * --------------------------------------------------------------------------
 */
static char _V_[] = "@(#)lbucket.c $Revision: 1.3 $";
static char _V_align[4-sizeof(_V_)%4] = {0*sizeof(_V_align)};


/**************************************************************************/
/*                          DESIGN DESCRIPTION                            */
/**************************************************************************/

/*
 * This file provides a general-purpose leaky bucket mechanism that can be
 * used, for example, to limit the number of events that are handled by a
 * particular process. A leaky bucket is an abstract object that is not
 * directly associated with a queue or any other object in the system.
 *
 * The leaky bucket descriptor (of type t_leaky_bucket) is typically allocated
 * as module-local data.
 *
 * Every leaky bucket has a fill level. When a user uses the bucket, a value
 * is added to the fill level. Every t microseconds, the value c is subtracted
 * from the fill level; however, the fill level is never allowed to fall below
 * zero. The value t may only take on a fixed set of values, corresponding to
 * the timer hooks available in the system. Typically, these values are 20 ms,
 * 50 ms, 100 ms, 1 s, 5 s, 1 minute, 1 hour, and 1 day; however, 1 day
 * corresponds to 86,400,000,000 us, which cannot be stored in a dword.
 *
 * The bucket size m is the maximum allowed fill level in the bucket.
 *
 * The difference between the maximum fill level and the current fill level is
 * called the 'credit'.
 */


/**************************************************************************/
/*                          TABLE OF CONTENTS                             */
/**************************************************************************/

/*
 * Callback function for draining the bucket:
 *              DrainLeakyBucket
 *
 * API functions:
 *              CreateLeakyBucket
 *              CreditLeft
 *              DestroyLeakyBucket
 *              ModifyLeakyBucket
 *              UseLeakyBucket
 *              UseLeakyBucketAfter
 */
```

```
/*****************************************************************************/
/*                              INCLUDE FILES                                */
/*****************************************************************************/

#include    "h_define.h"
#include    INCL_OLICOS_H
#include    "lbucket.h" /*INCL_LBUCKET_H*/
#include    INCL_HOOKS_H

/*****************************************************************************/
/*                      PRIVATE TYPES and DEFINITIONS                        */
/*****************************************************************************/


/*****************************************************************************/
/*                              PRIVATE DATA                                 */
/*****************************************************************************/

/* TBD: This must be declared elsewhere: */

extern t_hook_event timer_20ms_hooks;
extern t_hook_event timer_50ms_hooks;
extern t_hook_event timer_100ms_hooks;
extern t_hook_event timer_1sec_hooks;
extern t_hook_event timer_5sec_hooks;
extern t_hook_event timer_1min_hooks;
extern t_hook_event timer_1hour_hooks;
extern t_hook_event timer_1day_hooks;


/* TBD: This must be moved to custom.c: */

t_timer_hooks aTimerHooks[] = {
    { &timer_20ms_hooks,        20000UL },
    { &timer_50ms_hooks,        50000UL },
    { &timer_100ms_hooks,      100000UL },
    { &timer_1sec_hooks,      1000000UL },
    { &timer_5sec_hooks,      5000000UL },
    { &timer_1min_hooks,     60000000UL },
    { &timer_1hour_hooks, 3600000000UL },
    { NULL, 0 }
};


/*****************************************************************************/
/*                              EXPORTED DATA                                */
/*****************************************************************************/


/*****************************************************************************/
/*                             PRIVATE FUNCTIONS                             */
/*****************************************************************************/

/*=============================  DrainLeakyBucket  ============================
** Function called by the hook mechanism to drain the bucket
**---------------------------------------------------------------------------*/
STATIC void             /*RET Nothing */
DrainLeakyBucket(
dword bucket)           /*IN  Address of leaky bucket */
{
    t_leaky_bucket_request *pevReq;
    t_leaky_bucket *pBucket = (t_leaky_bucket*)bucket;

    critical_on();

    if (pBucket->currentLevel < pBucket->drainAmount)
        pBucket->currentLevel = 0;
    else
        pBucket->currentLevel -= pBucket->drainAmount;

    /* Is somebody waiting that now has sufficient credit */
    while ((pevReq = (t_leaky_bucket_request*)queue_peek(&pBucket->evQ))!=NULL) {
        dword realMax = pevReq->maxi==0 ? pBucket->maxLevel : pevReq->maxi;

        /* Is there now credit enough to service this request? */
        if (realMax >= pBucket->currentLevel + pevReq->amount) {
            /* We have enough credit */
            pBucket->currentLevel += pevReq->amount;

            (void)queue_get(&pBucket->evQ);      /* Remove pevReq from queue... */
            request_start(&pevReq->event_head); /* ...and send it */
        }
```

```
        else
            break;
    }
    critical_off();
}

/**************************************************************************/
/*                          EXPORTED FUNCTIONS                            */
/**************************************************************************/

/*========================== CreateLeakyBucket ===========================
** Create a leaky bucket.
** The argument t must correspond to one of the timer hooks configured
** in the system.
**-----------------------------------------------------------------------*/
t_return                            /*RET Error code */
CreateLeakyBucket(
t_leaky_bucket *pBucket,            /*IN  Address of leaky bucket descriptor */
dword c,                            /*IN  Drain amount every t us */
dword m,                            /*IN  Fill level maximum  */
dword t)                            /*IN  Time in us between drains */
{
    t_timer_hooks *pTh;

    /* Find a timer hook that can handle the time t */
    for (pTh=aTimerHooks; pTh->hook; pTh++)
        if (pTh->microsec==t) break;

    if (!pTh->hook) /* We didn't find a hook */
        return CC_RANGE_ERROR;

    /* Initialize the bucket */
    pBucket->drainAmount = c;
    pBucket->drainTime = t;
    pBucket->maxLevel = m;
    pBucket->currentLevel = 0;
    pBucket->pProcess = NULL;

    /* Set up the timer hook to handle the bucket */
    pBucket->hook = set_hook(pTh->hook, DrainLeakyBucket, (dword)pBucket);

    return CC_OK;
}

/*========================== CreditLeft ===================================
** Returns the amount of credit left in the bucket
**-----------------------------------------------------------------------*/
sdword                              /*RET Credit left */
CreditLeft(
t_leaky_bucket *pBucket)            /*IN  Address of leaky bucket descriptor */
{
    return pBucket->maxLevel - pBucket->currentLevel;
}

/*========================== DestroyLeakyBucket ==========================
** DestroyLeakyBucket prevents the timer from further decrementing the fill
** level. After this function has been called, there will be no external
** references to the leaky bucket, and the de-scriptor may be deallocated.
**-----------------------------------------------------------------------*/
void                          /*RET Nothing */
DestroyLeakyBucket(
t_leaky_bucket *pBucket)      /*IN  Address of leaky bucket descriptor */
{
    ASSERT(pBucket->hook!=NULL, pBucket); /* Primitive test that bucket is in use */
    ASSERT(pBucket->pProcess==NULL, pBucket);   /* We must not destroy
                                                   overload buckets before the
                                                   process is dead */

    clear_hook(&pBucket->hook);
}

/*========================== ModifyLeakyBucket ===========================
** Modifies the parameters of an existing leaky bucket. The current fill level
** in the bucket will be left untouched, even if it is greater than m.
**-----------------------------------------------------------------------*/
t_return                            /*RET Error code */
ModifyLeakyBucket(
t_leaky_bucket *pBucket,            /*IN  Address of leaky bucket descriptor */
dword c,                            /*IN  Drain amount every t us */
dword m,                            /*IN  Fill level maximum */
dword t)                            /*IN  Time in us between drains */
```

```
{
    t_timer_hooks *pTh;
    dword oldMaxLevel;
    t_leaky_bucket_request *pevReq;

    ASSERT(pBucket->hook!=NULL, pBucket); /* Primitive test that bucket is in use */

    /* Find a timer hook that can handle the time t */
    for (pTh=aTimerHooks; pTh->hook; pTh++)
        if (pTh->microsec==t) break;

    if (!pTh->hook) /* We didn't find a hook */
        return CC_RANGE_ERROR;

    critical_on();

    /* Reinitialize the bucket, but don't touch currentLevel */
    oldMaxLevel = pBucket->maxLevel;
    pBucket->drainAmount = c;
    pBucket->drainTime = t;
    pBucket->maxLevel = m;

    /* Set up the timer hook to handle the bucket */
    clear_hook(&pBucket->hook);
    pBucket->hook = set_hook(pTh->hook, DrainLeakyBucket, (dword)pBucket);


    /* If the maxLevel increased, we may now be able to service some waiting events */
    if (m > oldMaxLevel) {
        while ((pevReq = (t_leaky_bucket_request*)queue_peek(&pBucket->evQ))!=NULL) {
            dword realMax = pevReq->maxi==0 ? pBucket->maxLevel : pevReq->maxi;

            /* Is there now credit enough to service this request? */
            if (realMax >= pBucket->currentLevel + pevReq->amount) {
                /* We have enough credit */
                pBucket->currentLevel += pevReq->amount;

                (void)queue_get(&pBucket->evQ);     /* Remove pevReq from queue... */
                request_start(&pevReq->event_head); /* ...and send it */
            }
            else
                break;
        }
    }

    critical_off();

    return CC_OK;
}

/*============================= UseLeakyBucket ==================================
** UseBucket increments the fill level on the specified leaky bucket by the
** specified amount. It returns TRUE if the fill level can be incremented
** without the bucket overflowing. If the available credit is less than amount
** or if somebody else is waiting for the bucket to drain, the fill level is
** not incremented and the function returns FALSE.
**
** If the argument maxi is greater than 0, it specifies a fill level maximum
** to be used instead of the value specified when the bucket was created.
**
** The argument pevReq may be NULL or the address of a user-defined event. If
** it is not NULL, it is assumed to be the address of an event. In this case
** the event will be returned to the calling process when the desired credit
** is available. When the event is returned, the fill level has already been
** incremented by the desired amount.
**-----------------------------------------------------------------------------*/
bool                                /*RET Success? */
UseLeakyBucket(
t_leaky_bucket *pBucket,            /*IN  Address of leaky bucket descriptor */
dword amount,                       /*IN  Usage amount */
dword maxi,                         /*IN  Alternative maximum (or 0 for default) */
t_leaky_bucket_request *pevReq)     /*IN  Leaky bucket request event */
{
    dword realMax;

    ASSERT(pBucket->hook!=NULL, pBucket); /* Primitive test that bucket is in use */

    critical_on();

    realMax = maxi==0 ? pBucket->maxLevel : maxi;

    /* Test if the bucket will overflow or if somebody is already
       waiting for the bucket */
```

```
    if (realMax >= pBucket->currentLevel + amount && queue_empty(&pBucket->evQ)) {
        /* No overflow and nobody waiting */
        pBucket->currentLevel += amount;

        critical_off();
        return TRUE;
    }

    /* Overflow or somebody waiting */
    if (pevReq) {
        pevReq->amount = amount;
        pevReq->maxi = maxi;
        queue_put(&pBucket->evQ, &pevReq->event_head.resource_head);
    }

    critical_off();
    return FALSE;
}

/*=========== UseLeakyBucketAfter ==========
** This is a special version of UseLeakyBucket. It is called _after_ the
** resource it monitors has been used. Therefore it never fails, but it does
** return an indication of whether an overflow occured or not.
**
** This function is primarily intended to be used with process overflow
** dectection.
**----------*/
bool                            /*RET Success? */
UseLeakyBucketAfter(
t_leaky_bucket *pBucket,        /*IN  Address of leaky bucket descriptor */
dword amount,                   /*IN  Usage amount */
dword maxi)                     /*IN  Alternative maximum (or 0 for default) */
{
    dword realMax;
    bool boNoOverflow;

    ASSERT(pBucket->hook!=NULL, pBucket); /* Primitive test that bucket is in use */

    critical_on();

    realMax = maxi==0 ? pBucket->maxLevel : maxi;
    pBucket->currentLevel += amount;
    boNoOverflow = (realMax >= pBucket->currentLevel);

    critical_off();
    return boNoOverflow;
}
```

```
#undef  INCL_LBUCKET_H
#define INCL_LBUCKET_H "empty.h"
/*
****************************** LBUCKET.H ********************************
 *
 *                        Copyright (c) 1998
 *                           OLICOM A/S
 *                            Denmark
 *
 *                       All Rights Reserved
 *
 *
 * This source file is subject to the terms and conditions of the
 * OLICOM Software License Agreement which restricts the manner
 * in which it may be used.
 *
 * ------------------------------------------------------------------------
 *
 * Description: Leaky bucket definitions
 *
 * ------------------------------------------------------------------------
 *
 * $Log: lbucket.h,v $
 * Revision 1.3  1998/06/15 12:31:59  cto
 * CreditLeft now returns sdword rather than sword
 *
 * Revision 1.2  1998/06/15 11:23:15  cto
 * Added function UseLeakyBucketAfter
 *
 * Revision 1.1  1998/06/04 10:43:06  cto
 * Addition of leaky bucket files lbucket.c, lbucket.h
 *
 *
 *
 * ------------------------------------------------------------------------
 */
/*  @(#)lbucket.h $Revision: 1.3 $ */

/***************************************************************************/
/*                                                                         */
/*                           INCLUDE FILES                                 */
/***************************************************************************/

#include    INCL_HOOKS_H

/***************************************************************************/
/*                                                                         */
/*                 EXPORTED TYPES and DEFINITIONS                          */
/***************************************************************************/

/* t_timer_hooks represents a relationship between a timer hook and the time
 * interval it represents.
 */
typedef struct timer_hooks {
    t_hook_event      *hook;        /* The hook */
    dword             microsec;     /* The corresponding time in us */
} t_timer_hooks;


/* The file custom.c must implement the array aTimerHooks, which contains the
 * mapping between timer values and the corresponding timer hook addresses.
 * The last entry must have hook==NULL.
 */
extern t_timer_hooks aTimerHooks[];

/* t_leaky_bucket_request is an event type that is used when a process
 * requests notification when credit becomes available.
 */
typedef struct leaky_bucket_request {
    t_event_head      event_head;  /* Standard event structure */
    dword             amount;      /* Requested credit */
    dword             maxi;        /* Maximum fill level */
} t_leaky_bucket_request;


/* t_leaky_bucket is the leaky bucket descriptor */

typedef struct leaky_bucket {
    t_hook          *hook;          /* Timer hook handling this bucket */
    dword           drainAmount;    /* Drain amount every t us */
    dword           drainTime;      /* Time in us between drains */
    dword           maxLevel;       /* Fill level maximum */
    dword           currentLevel;   /* Current fill level */
```

```
    t_queue_head evQ;              /* Queue of events that are to be sent when
                                      there is room in the bucket */
    t_process    *pProcess;        /* Address of process for which this bucket
                                      is an overload bucket. This field is
                                      NULL if this is not an overload bucket. */
} t_leaky_bucket;

/*****************************************************************************/
/*                              EXPORTED DATA                                */
/*****************************************************************************/

/*****************************************************************************/
/*                            EXPORTED FUNCTIONS                             */
/*****************************************************************************/

/*========================= CreateLeakyBucket ================================
** Create a leaky bucket.
** The argument t must correspond to one of the timer hooks configured
** in the system.
**--------------------------------------------------------------------------*/
t_return                           /*RET Error code */
CreateLeakyBucket(
t_leaky_bucket *pBucket,           /*IN  Address of leaky bucket descriptor */
dword c,                           /*IN  Drain amount every t us */
dword m,                           /*IN  Fill level maximum  */
dword t);                          /*IN  Time in us between drains */

/*============================ CreditLeft ====================================
** Returns the amount of credit left in the bucket
**--------------------------------------------------------------------------*/
sdword                             /*RET Credit left */
CreditLeft(
t_leaky_bucket *pBucket);          /*IN  Address of leaky bucket descriptor */

/*======================== DestroyLeakyBucket ================================
** DestroyLeakyBucket prevents the timer from further decrementing the fill
** level. After this function has been called, there will be no external
** references to the leaky bucket, and the de-scriptor may be deallocated.
**--------------------------------------------------------------------------*/
void                       /*RET Nothing */
DestroyLeakyBucket(
t_leaky_bucket *pBucket);  /*IN  Address of leaky bucket descriptor */


/*========================= ModifyLeakyBucket ================================
** Modifies the parameters of an existing leaky bucket. The current fill level
** in the bucket will be left untouched, even if it is greater than m.
**--------------------------------------------------------------------------*/
t_return                           /*RET Error code */
ModifyLeakyBucket(
t_leaky_bucket *pBucket,           /*IN  Address of leaky bucket descriptor */
dword c,                           /*IN  Drain amount every t us */
dword m,                           /*IN  Fill level maximum */
dword t);                          /*IN  Time in us between drains */

/*========================== UseLeakyBucket ==================================
** UseBucket increments the fill level on the specified leaky bucket by the
** specified amount. It returns TRUE if the fill level can be incremented
** without the bucket overflowing. If the available credit is less than amount
** or if somebody else is waiting for the bucket to drain, the fill level is
** not incremented and the function returns FALSE.
**
** If the argument maxi is greater than 0, it specifies a fill level maximum
** to be used instead of the value specified when the bucket was created.
**
** The argument pevReq may be NULL or the address of a user-defined event. If
** it is not NULL, it is assumed to be the address of an event. In this case
** the event will be returned to the calling process when the desired credit
** is available. When the event is returned, the fill level has already been
** incremented by the desired amount.
**--------------------------------------------------------------------------*/
bool                               /*RET Success? */
UseLeakyBucket(
t_leaky_bucket *pBucket,           /*IN  Address of leaky bucket descriptor */
dword amount,                      /*IN  Usage amount */
dword maxi,                        /*IN  Alternative maximum (or 0 for default) */
```

```
t_leaky_bucket_request *pevReq);/*IN  Leaky bucket request event */

/*---------------------- UseLeakyBucketAfter ----------------------
** This is a special version of UseLeakyBucket. It is called _after_ the
** resource it monitors has been used. Therefore it never fails, but it does
** return an indication of whether an overflow occured or not.
**
** This function is primarily intended to be used with process overflow
** deccection.
**----------------------------------------------------------------------*/
bool                                /*RET Success? */
UseLeakyBucketAfter(
t_leaky_bucket *pBucket,            /*IN  Address of leaky bucket descriptor */
dword amount,                       /*IN  Usage amount */
dword maxi);                            /*IN  Alternative maximum (or 0 for default) */
```

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [ ] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [ ] FADED TEXT OR DRAWING
- [x] BLURRED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [ ] COLOR OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [ ] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [ ] OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.